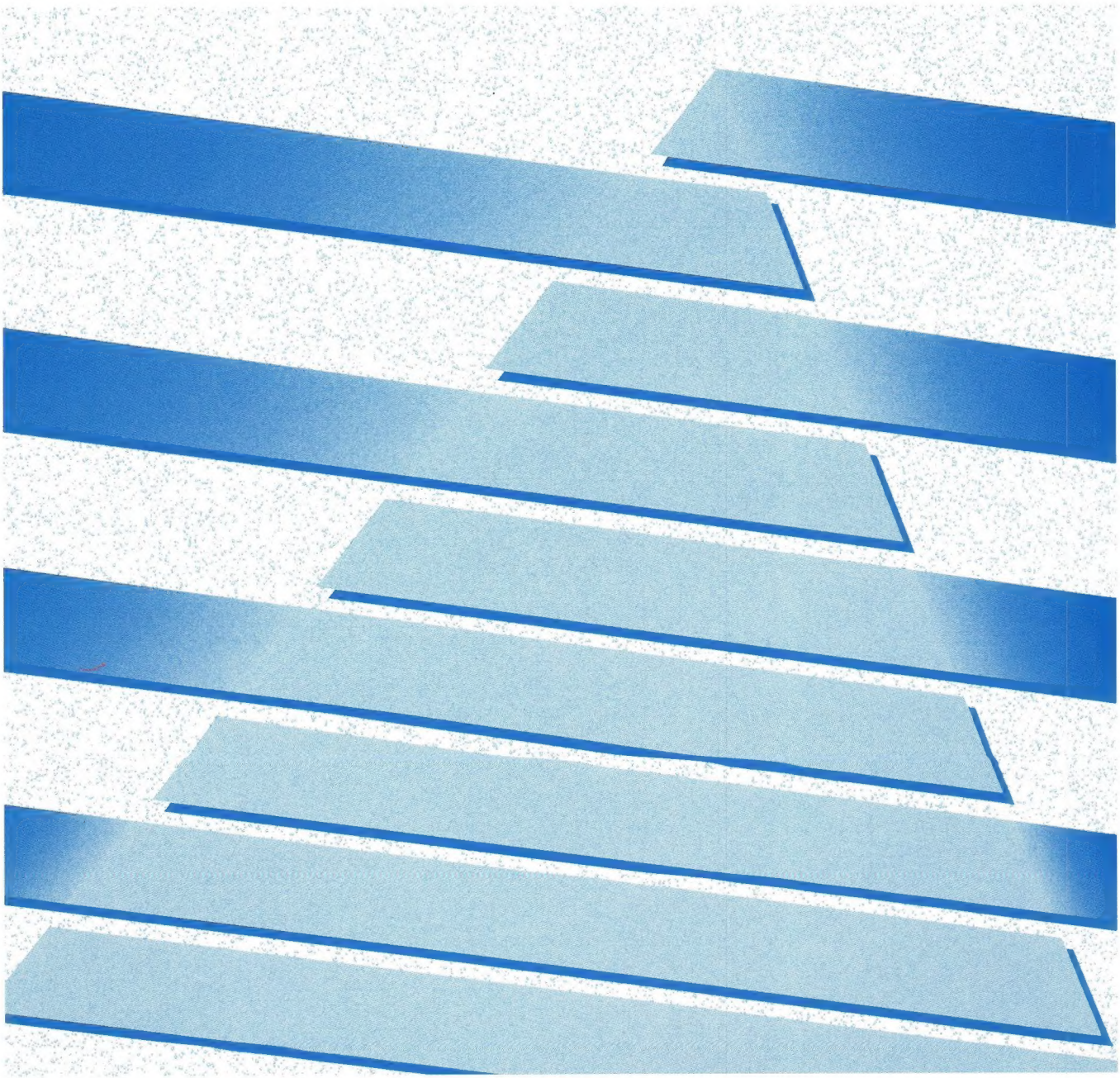
ALLEN-BRADLEY

ControlView™ Event Detector

(Cat. No. 6190-EVD)

User Manual



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, the Allen-Bradley Company, Inc. does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley Publication SGI-1.1, "Safety Guidelines for the Application, Installation and Maintenance of Solid State Control" (available from your local Allen-Bradley office) describes some important differences between solid-state equipment and electromechanical devices which should be taken into consideration when applying products such as those described in this publication.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss.

Attentions help you:

- identify a hazard
- avoid the hazard
- recognize the consequences

Important: Identifies information that is especially important for successful application and understanding of the product.

Summary of Changes

Changes from Release 2.0 to 3.0

The following changes have been made to the Event Detector option and the Event Detector User Manual since release 2.0:

For information on this new feature:	Refer to:	The feature appeared in:
Installation instructions for the Event Detector option have been moved to the *ControlView Installation Manual.*	ControlView Installation Manual	software release 3.0

Preface

How To Use This Manual

This manual describes the features and capabilities of the Event Detector option of the ControlView™ System.

Conventions Used in This Manual

This manual follows the print conventions outlined in the *ControlView Core User Manual.*

Audience

Since the Event Detector is a part of ControlView, you should be familiar with ControlView and have the *ControlView Core User Manual* available for reference. A complete list of related publications is contained in that manual.

Using the Event Editor

Chapter 1

Defining an Expression

Chapter 2

Event Detector Command

Appendix A

Index

Chapter 1

Using the Event Editor

Introduction

The Event Detector option enables you to monitor specific conditions occurring on the plant floor and to respond to those conditions or *events* when they take place.

Events are defined as expressions, that is, as equations containing tag values, mathematical operations, IF-THEN-ELSE logic and other built-in ControlView functions. When an event occurs, a ControlView command is generated causing a corresponding action to occur. This action could, for example:

- initiate a Data Logger model
- display a help window
- start a C-Toolkit program
- respond to an unsolicited message from a programmable controller (PLC®)

There are two parts to the Event Detector:

- the Event Editor, which you use to define events
- the Event Detector, which monitors the events and triggers the specified actions

With the Event Editor you can:

- define events based on expressions
- specify the evaluation rate of the events
- enable or disable events individually
- optionally verify the validity of all tag references

Event files can contain up to 300 events and can refer to up to 300 tags.

This chapter describes how to use the Event Editor to create, edit, and enable or disable events. It does not describe the expression language that is used to produce the event's actual function. If you do not know how to use the expression language, refer to Chapter 2, *Defining an Expression.*

Like all ControlView options, Event Detector can be managed and run both through ControlView's Setup and Actions Menus and through the Command Line. For details on the commands used with Event Detector, refer to Appendix A, *Event Detector Commands* at the end of this manual.

At the time of this release, string tags are not supported by this option.

Defining Events

To define or edit events:

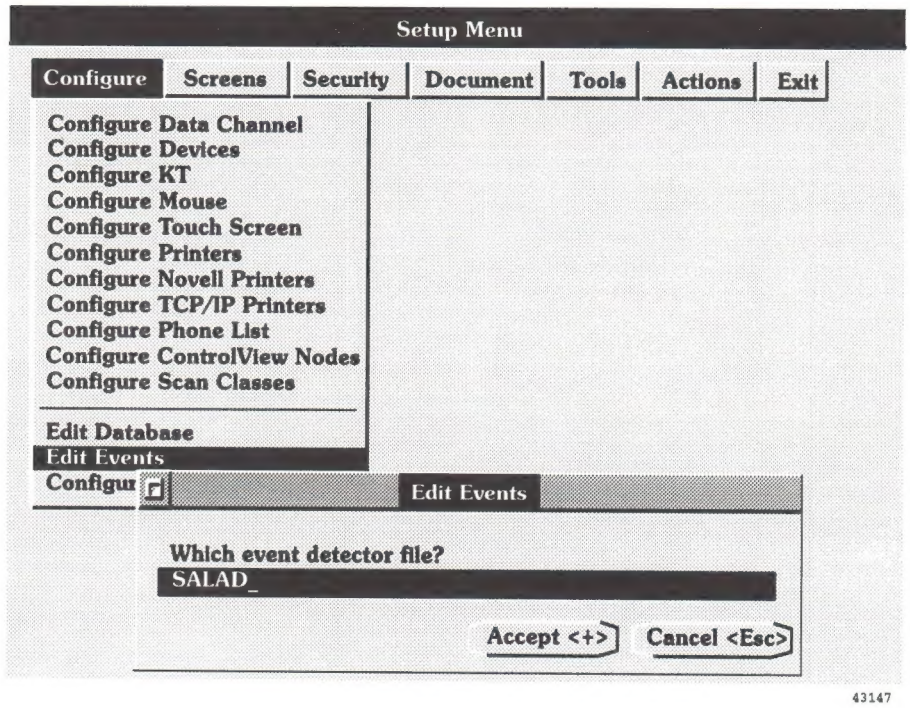
1. Load the database that contains the tags you will refer to in your expression. (If the database is loaded but doesn't contain some of the tags included in the expression, then go to *Setup* and set autoverification to *No*.)

2. Choose *Edit Events* under *Configure* in the Setup menu. You are prompted to name an event file.

3. To edit an existing file, type in its name; to create a new file, type in a new file name; or leave the field blank to use a default file called EVENTS.

   If you cannot remember the name of the event file, go to the Actions Menu and choose *Start Event Detector* under Start/Stop. A popup list of available event files is displayed.

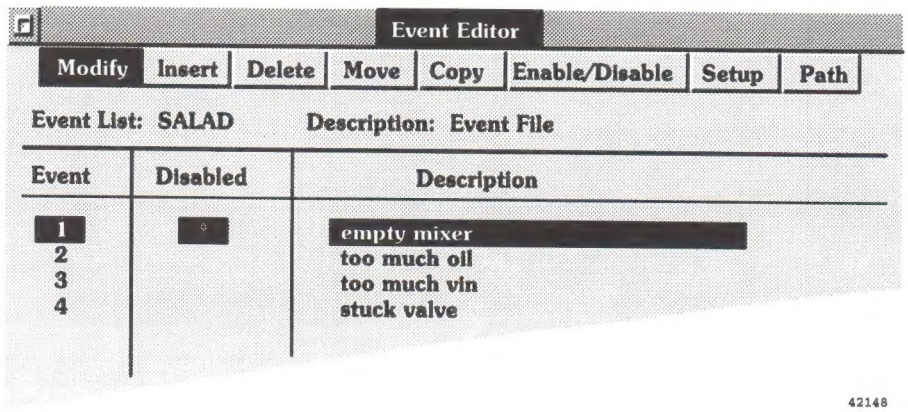
Figure 1.1
Starting the Event Editor



4. The Event Editor opens. If there are no events to edit, choose *Insert* to create new events.

In the following sections, the menu items are described in left-to-right order.

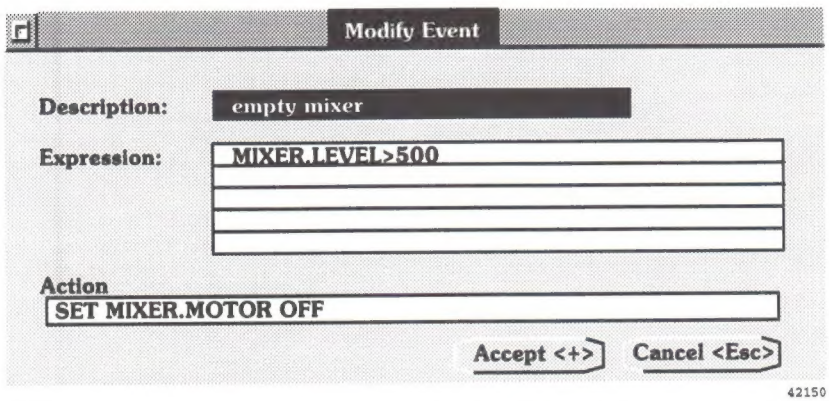
Figure 1.2
The Event Editor Window



Modify

To edit an event definition, highlight the event and choose *Modify*, or double-click on the event. You can modify it in the Modify Event window.

Figure 1.3
Modify Event Window



The information appearing in any of the three fields in this window can be changed. For details on the *Description*, *Expression* and *Action* fields, see the description of Insert, below.

Save the changes with the *Accept* button or the + key.

Insert

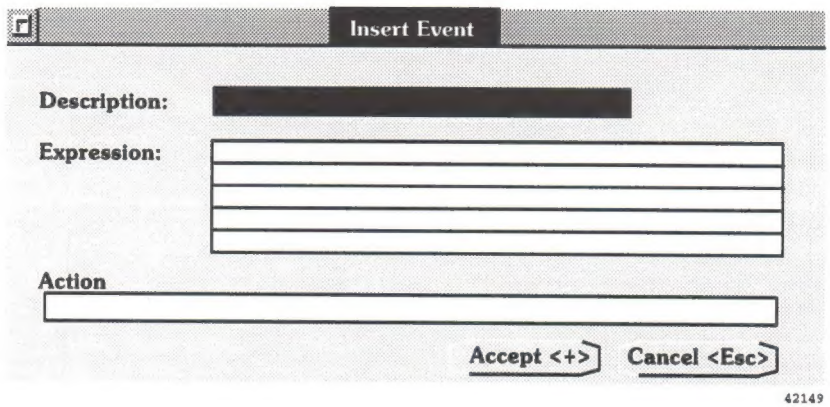
To create a new event:

1. Determine where the new event will go by highlighting an event in the Event Editor. The new event will appear below the selected one.

2. Choose *Insert* to add a new event. The Insert Event window opens.

Important: Events are processed in the order they are listed. So, if a particular action depends on another action being performed first, list their associated events in the same sequence that the actions will occur.

Figure 1.4
Insert Event Window



- Description

  Enter a brief description to document the event's function. This description is listed in the Event Editor's main window.

- Expression

  Enter an expression to specify the conditions that will trigger an action. *When the expression changes from FALSE on the previous evaluation to TRUE on the current evaluation, the Event Detector performs the associated action.* For details on how to define an expression, refer to Chapter 2, *Defining an Expression.*

- Action

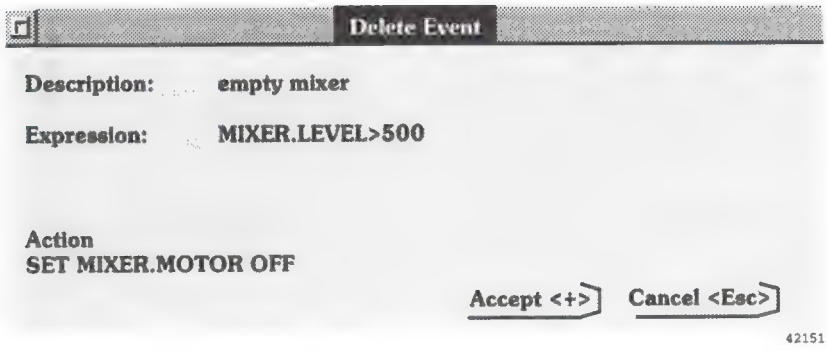
  Type in a command, macro, or symbol that will run when the expression goes from FALSE to TRUE. For more information on macros and symbols, see the *ControlView Core User Manual,* Chapter 7, *Customizing the System.*

Save the new event with the *Accept* button or the + key.

Delete

To erase an event, highlight it and choose *Delete*. The event's definition appears in the window illustrated in Figure 1-5. Confirm the deletion with the *Accept* button or the + key.

Figure 1.5
Delete Event Window



Move

Events are processed in the order they appear in the event list. So, if the second event depends on the result of the first event, ensure that event 1 is listed earlier.

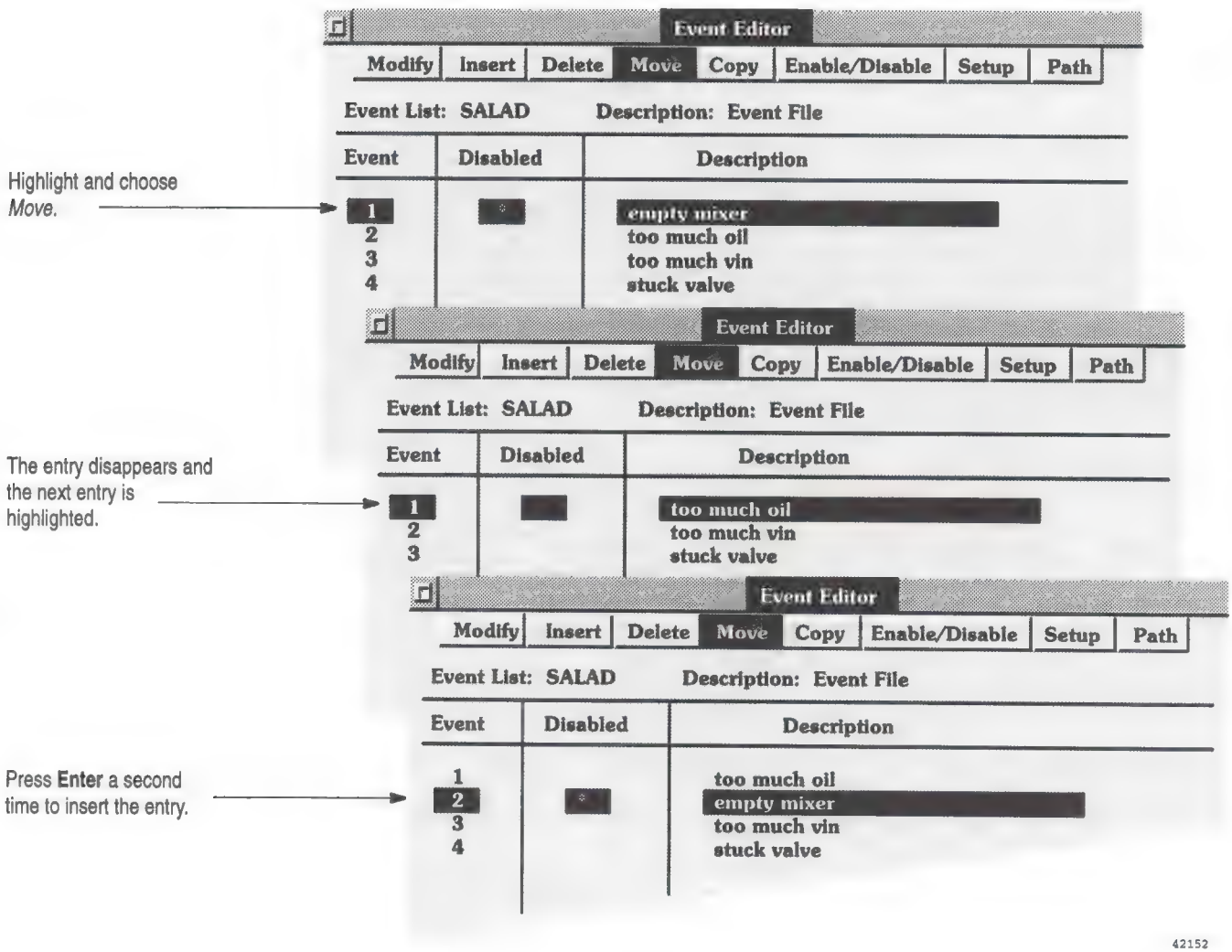
To change the order:

1. Highlight an event and choose *Move*. The highlighted event disappears from the list and a blue flashing highlight bar appears on the next event in the Event Editor.

2. Highlight a new position and press **Enter** (or double-click on the destination). The event will reappear below the highlighted entry.

Important: To move an event to the top of the list, move it below the top entry, and then move the top entry down one position.

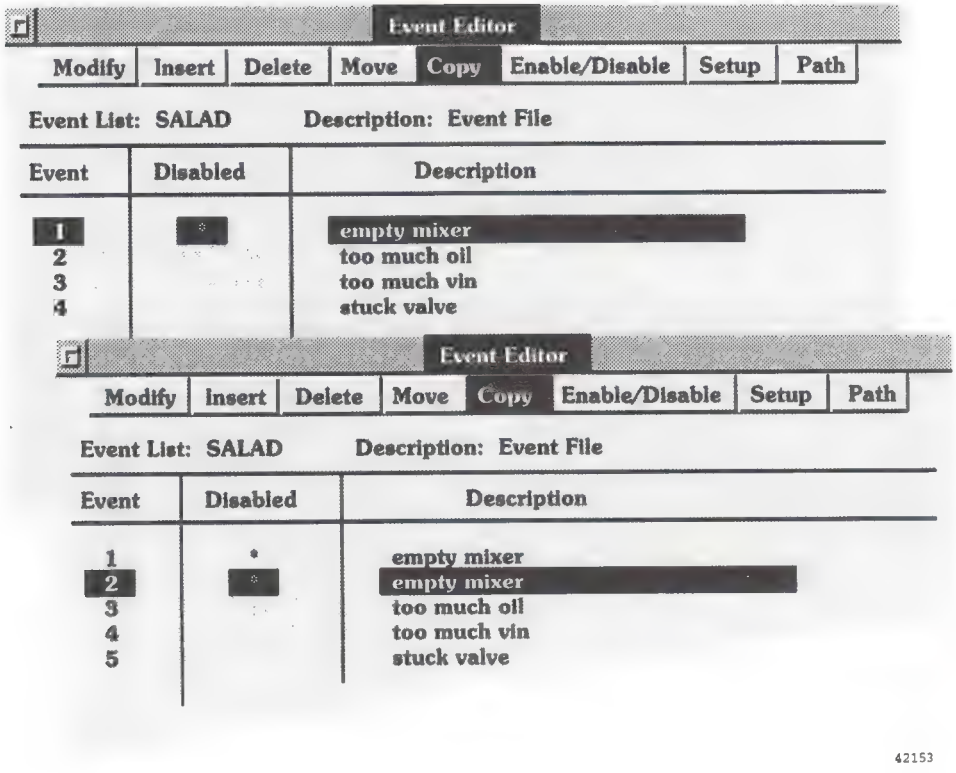
Figure 1.6
Moving an Event



Copy

To duplicate an event, highlight it and choose *Copy*. An identical entry appears below the original. You can then move and modify the copy.

Figure 1.7
Copying an Event



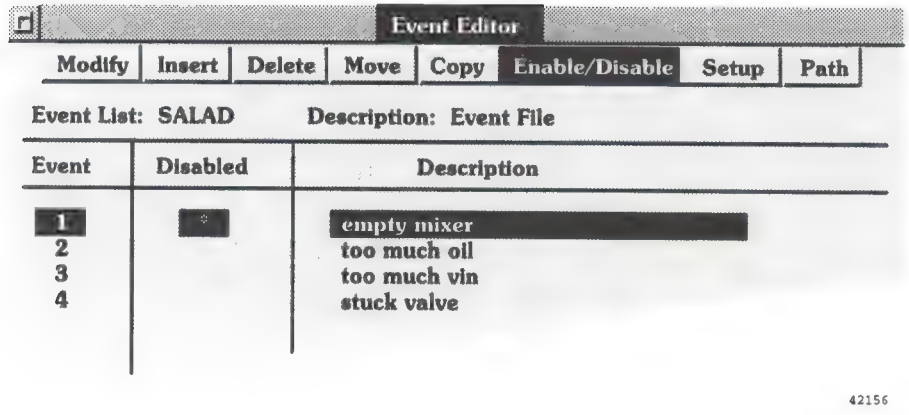
Enable/Disable

Disabling an event turns it off so the Event Detector won't perform the specified action when the expression becomes TRUE. Enabling an event turns a disabled event back on.

To switch the event to and from the disabled and enabled states, highlight the event and choose *Enable/Disable*.

Disabled events listed in the Event Editor are identified by an asterisk in the Disabled column.

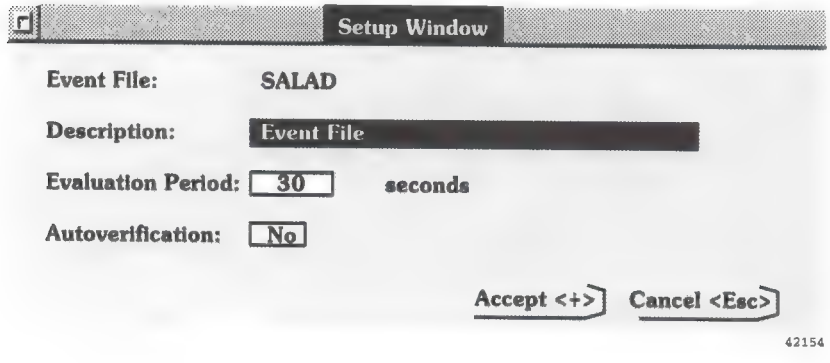
Figure 1.8
Asterisk Shows That Event Is Disabled



Setup

Use *Setup* to define the characteristics of the event file. When you choose *Setup*, the following window appears.

Figure 1.9
Setup Window



- Event File

  This field displays the event file being edited.

- Description

  Enter a brief description of the file to identify it. This description is used only for documentation purposes.

- Evaluation Period

  The Event Detector continually cycles through the list of events evaluating each expression and triggering an action when the specified condition or event occurs. The frequency of the cycle, or Evaluation Period, ranges from 0 to 99,999,999 seconds.

  An Evaluation Period of 0 means the Event Detector will cycle through the list as fast as possible without pausing. However, such speed can impair system performance since the computer will spend a lot of time computing the expressions.

  If your expressions rely on tag values, remember that the tags in the current value database are updated according to specific scan class rates. Set the evaluation period for the event file to be equal to or slower than the scan rate for those tags. If you set the rate faster, the expressions will be recalculated using values that haven't changed, and the system will do needless processing. For more information on scan classes, see *ControlView Core User Manual*, Chapter 2, *The Setup Menu.*

  To complete the Evaluation Period field, specify, in seconds, how often the Event Detector should cycle through the list of events.

  If you define events using any of the Time functions (TIME(), BEFORE_TIME(), AFTER_TIME(), or INTERVAL()), then the Evaluation Period is not tied to the scan rates alone. For more information on Expressions see Chapter 2, *Defining an Expression.*

- Autoverification

  When autoverification is on, the database is checked for the tags used in the expression field. If a tag used in an expression is not in the database, an error message appears.

  Important: Autoverification does not verify the existence of any tags used in the Action field.

  Choose *YES* to turn autoverification on, *NO* to turn it off.

  Important: When you start the Event Editor from the menu, autoverification is always turned off, regardless of the setting for this field. Autoverification can only be used when the editor is loaded with the EVENT command.

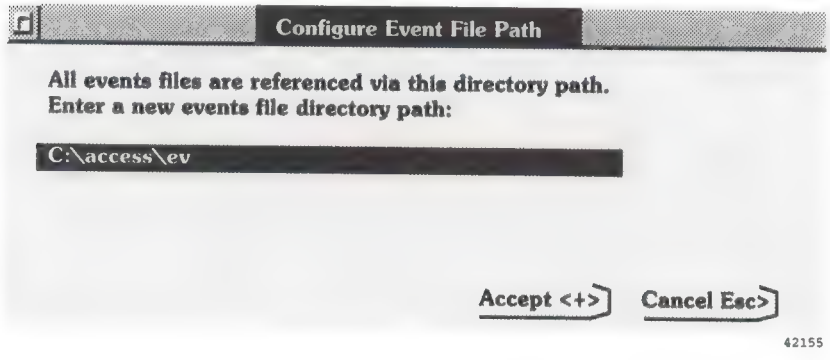
Path

Specify the DOS pathname where the event files are located. The default directory is C:\ACCESS\EV where C is the drive where ControlView is installed.

Important: Whenever you configure a pathname, be sure to start with the drive letter. This is absolutely essential when running ControlView in a multi-drive environment.

Changing the path of an existing event file, copies the event file to the new directory. The event file also remains in the previous path. If the file already exists in the new directory, its contents will be overwritten.

Figure 1.10
Configure Event File Path Window



Start the Event Detector

Once the events are defined, the Event Detector may be started. To load the Detector, go to Start/Stop in the Actions menu and choose *Start Event Detector.* A list of the event files will appear. Choose the file you wish to run.

Important: Only one event file can be run at a time.

You can use autoverification, to check that the tags listed in the expressions actually exist in the database. When you open the Event Editor from the menu, autoverification is turned OFF, which allows you to set up the expressions before your database has been completed.

To use autoverification, the database must be loaded before you call up the Event Editor.

Important: Tag values are only read in at the beginning of the Event Detector evaluation cycle. If any tags change during the evaluation cycle, the new tag value will not be read until the next evaluation cycle.

Stop the Event Detector

To stop the Event Detector, choose *Stop Event Detector* under Start/Stop in the Actions Menu.

The Event Detector can also be run from the command line. See Appendix A, *Event Detector Commands* for more information.

Chapter 2

Defining an Expression

Defining an Expression

Sometimes the data gathered from remote drivers is only meaningful when you:

- combine it with other values
- compare it to other values
- create a cause - effect relationship with other values

Expressions let you create mathematical or logical combinations of data that return more meaningful values.The term refers to an entire equation whereas a segment of an expression is called a *statement.*

Expressions can be built from:

- tag values
- constants
- mathematical, relational, logical, and bitwise operators
- built-in functions
- IF-THEN-ELSE program logic

Important: All expressions return floating point values.

Example: Expressions vs Statements

```
(tag1 * tag2) AND (tag3 / 2)
```

is an expression

```
(tag3 / 2)
```

is a statement

Tag Values

A tag can be included as part of an expression, or can stand alone as the entire expression.

Constants

Using the number 123.45 as an example, a constant could have any of the following formats:

- integer (123)
- floating point value (123.45)
- scientific notation (1.2345E2)

Arithmetic Operators

The table below shows the arithmetic operators.

Table 2.A
The Arithmetic Operators

Symbol	Operator	Example
+	addition	tag1 + tag2
–	subtraction	tag1 – tag2
*	multiplication	tag1 * tag2
/	division	tag1 / tag2
MOD, %	modulus (remainder)	tag1 % tag2
**	exponent	tag1 ** tag2

The modulus operator is the remainder of one number divided by another. For example, the remainder of 13 divided by 5 is 3; so 13 % 5 = 3.

Important: Expressions which attempt to divide a number by zero are ignored. No error message is given. Be sure that any tag value which is used as a divisor cannot at some point have a value of zero.

Examples: Arithmetic Operators

For these examples, tag1 = 5 and tag2 = 7.

```
tag1 + tag2
```
returns a value of 12

```
tag1 * tag2
```
returns a value of 35

```
tag1 - tag2
```
returns a value of –2

```
tag1 / tag2
```
returns a value of 0.71

```
tag1 MOD tag2
```
returns a value of 5

```
tag1 ** tag2
```
returns a value of 78125

Important: Operator precedence is covered later in this chapter.

Relational Operators

Relational operators compare two values to provide a true or false result. If the statement is true, a value of 1 is returned. If false, 0 is returned. There are six relational operators; each has two different symbols.

Table 2.B
The Relational Operators

Symbols	Operator	Example
EQ, =	equal	tag1 = tag2
NE, <>	not equal	tag1 <> tag2
LT, <	less than	tag1 < tag2
GT, >	greater than	tag1 > tag2
LE, <=	less than or equal to	tag1 <= tag2
GE, >=	greater than or equal to	tag1 >= tag2

Examples: Relational Operators

For these examples, tag1 = 5 and tag2 = 7.

```
tag1 > tag2
```
is false, so the expression returns a 0

```
tag1 LE tag2
```
is true, so the expression returns a 1

```
tag1 = 5
```
is true, so the expression returns a 1

Logical Operators

Logical operators determine the validity of one or more statements. The operators return a value of 1 if the expression is true, or a value of 0 if false. There are three logical operators: AND, OR, and NOT.

- AND returns a value of 1 if the statement to the right and to the left of the operator are *both* true
- OR returns a value of 1 if *either* the statement to the left or to the right of the operator is true
- NOT reverses the logical value of the statement it operates on

Important: Any statement which evaluates to a non-zero value is regarded as true. For example, the statement tag1 will be false if the value of tag1 is 0 and true if tag1 has any other value.

Table 2.C
The Logical Operators

Symbols	Operator	Example
AND, &&	and	(tag1 < tag2) AND (tag1 = 5)
OR, ||	or	(tag1 = 5) OR (tag1 = 10)
NOT	negation	NOT(tag1 > tag2)

Examples: Logical Operators

For these examples, tag1 = 5 and tag2 = 7.

```
(tag1 < tag 2) AND (tag1 = 5)
```
returns a 1 since both statements are true

```
tag1 && tag2
```
returns a 1 since both tag1 and tag2 are non-zero (true)

```
(tag1 > tag2) OR (tag1 = 5)
```
returns a value of 1 since *tag1 = 5* is true

```
NOT(tag1 < tag2)
```
tag1 < tag2 is true and would return a 1 but the NOT operator reverses the logical value, so 0 is returned

Important: The parentheses are essential in the above expressions.

Bitwise Operators

Bitwise operators let you to examine and manipulate individual bits within a value. These operators can only be applied to integers, not floating point numbers.

Table 2.D
The Six Bitwise Operators

Symbol	Operator	Example
&	AND	tag1 & 07
|	inclusive OR	tag2 | tag1
^	exclusive OR (XOR)	tag1 ^ 01
>>	right shift	tag1 >> 1
<<	left shift	tag1 << 2
~	complement	~ tag1

The bitwise operators &, |, and ^ compare two integers or tags on a bit by bit basis. The >>, <<, and ~ operators manipulate a single integer or tag.

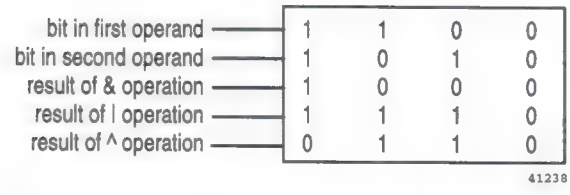
The **bitwise AND (&)** returns an integer with a bit set to 1 if both of the corresponding bits in the original numbers are 1. Otherwise, the resulting bit is 0.

The **bitwise inclusive OR (|)** returns an integer with a bit set to 1 if either or both of the corresponding bits in the original numbers are 1. If both bits are 0, the resulting bit is 0.

The **bitwise exclusive OR (^)** returns an integer with a bit set to 1 if either of the corresponding bits in the original numbers is 1. If both bits are 1 or both are 0, the resulting bit is 0.

The bitwise operators &, |, and ^ are illustrated below:

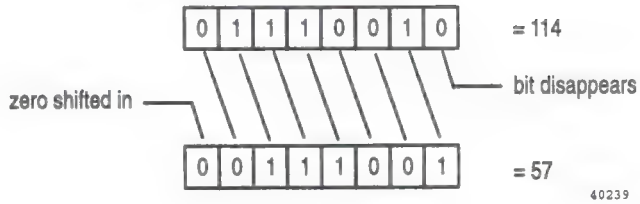
Figure 2.1
The &, |, and ^ Bitwise Operators



The **shift right operator (>>)** shifts the bits within the left operand by the amount specified in the right operand. The bit on the right disappears.

Figure 2.2 shows the result of the expression 114 >> 1 = 57.

Figure 2.2
The Shift Right Operator



Either a 0 or a 1 is shifted in on the left, depending on whether the integer is signed or unsigned. With unsigned integers, 0 is always shifted in on the left; with signed integers, a 0 is shifted in when the number is positive (i.e. the leftmost bit, the sign bit, is 0), and a 1 is shifted in when the number is negative (i.e. the leftmost bit, the sign bit, is 1). In other words, with signed integers, the sign of the number is always maintained.

The **shift left operator** (<<) shifts the bits within the left operand by the amount specified in the right operand. The bit on the left disappears and a 0 is always shifted in on the right.

The **bitwise complement operator** (~) reverses every bit within the number, so that every 1 bit becomes a 0 and vice versa.

Examples: Bitwise Operators

For these examples tag1 = 5 (binary 0101), tag2 = 2 (binary 0010)

```
tag1 & tag2
```
returns 0 (binary 0000)

```
tag1 | tag2
```
returns 7 (binary 0111)

```
tag1 ^ tag2
```
returns 7 (binary 0111)

```
tag1 >> 1
```
returns 2 (binary 0010)

```
tag1 << 1
```
returns 10 (binary 1010)

```
~ tag1
```
returns 10 (binary 1010)

Built-in Functions

There are three types of built-in functions:

- functions involving tags
- functions involving the time
- functions involving file operations

These functions return floating point values.

Many of the functions check for specific true/false conditions. Such functions return 1 if the condition is true, and 0 if the condition is false. For instance, the COMM_ERR() function returns 1 if the specified tag has a communication error.

Tag Functions

The following built-in functions examine the status of a tag:

Table 2.E
Built-in Functions

This function:	returns this value:
SQRT(*tag*)	the square root of the tag (or of a constant).
COMM_ERR(*tag*)	TRUE if a read or write operation for the specified tag produced a communication failure.
ALM_SUPPRESS(*tag*)	TRUE if the tag's alarms are suppressed.
ALM_FAULT(*tag*)	TRUE if there has been an alarm fault for the specified tag.
ALM_SEVERITY(*tag*)	the severity of the alarm—a value between 1 and 8, or 0 if the tag isn't in alarm.
ALM_LEVEL(*tag*)	the alarm level for the tag: a value between 1 and 8, or 0 if the tag isn't in alarm.
ALM_ACK(*tag*)	TRUE if the tag's alarm has been acknowledged.
ALM_IN_ALARM(*tag*)	TRUE if the tag is in alarm.

Important: The alarm (ALM) functions will only be relevant if the Alarming option is installed.

Examples: Tag Functions

```
ALM_IN_ALARM(vessel3.TIC3.pv)
```
returns 1 if the tag is in alarm or 0 if it isn't

```
SQRT(vessel3.TIC2.pv)
```
returns the square root of the tag value

```
SQRT(25)
```
returns a value of 5

Time Functions

The following built-in functions examine the system time. These functions use the *time* or *interval* parameters.

Important: the time or interval parameter must be surrounded by quotes.

The *time* parameter can include the following options:

- day of week [Sun, Mon, Tue, Wed, Thu, Fri, or Sat]
- month [Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec]
- date [1 – 31]
- year [1980 – 2100]
- hour of day [00: – 23:]
- minute [:00 – :59]

It doesn't matter what order you list these options in.

Example: Time Parameter

The following all represent the same date and time, and are valid time parameters:

- `"fri aug 18 1989 17:00"`
- `"fri aug 18 1989 17: :00"`
- `":00 aug 18 fri 1989 17:"`

Important: The validity of the date is not checked, so if Aug 18 1989 is *not* a Friday, this error will not be detected.

Example: Incomplete Time Parameter

The following are valid examples of incomplete time parameters:

- `"17:00"`
  means once a day at 5:00 pm
- `" :30 "`
  means once an hour, on the half hour
- `" fri 17:"`
  means 5:00 pm each Friday

The *interval* parameter has this format:

<number> <units>

where *<units>* is one of:

- sec
- min
- hour
- day
- week
- mon
- year

When defining the interval parameter, remember that the Evaluation Period, defined in the Setup screen, determines how frequently events are evaluated. Do not set the interval time to a duration shorter than the Evaluation Period.

Table 2.F
The Time Functions

This function:	returns this value:
TIME("*time*")	TRUE if the time specified is the current time.
BEFORE_TIME("*time*")	TRUE if the expression is evaluated before the specified time.
AFTER_TIME("*time*")	TRUE if the expression is evaluated after the specified time.
INTERVAL("*interval*")	TRUE if the specified time interval has elapsed - the interval timer is started when Event Detector is started.

Examples: Time and Interval Functions

```
TIME(" sun  feb  18  1990  14:30")
```

returns 1 if it is exactly 2:30 pm on Sunday Feb 18, 1990; otherwise 0 is returned.

Important: If the Evaluation Period is greater than 60 seconds, this function may never be evaluated as true.

```
AFTER_TIME("sun feb 18 1990  14: :30")
```

returns 1 the first time the expression is evaluated after 2:30 pm on Sunday Feb 18, 1990.

The expression will continue returning 1 from this point on, but since an action is triggered only when the expression changes from FALSE to TRUE, the action will be triggered only once.

```
BEFORE_TIME("Feb 18 1990")
```

returns 1 if it is not yet Feb 18, 1990; if the date is on or after Feb 18, 1990, 0 is returned.

```
INTERVAL("1 min")
```

returns 1 if a minute has elapsed since the expression last returned a 1.

Important: if the Evaluation Period is set for a longer duration, say 2 minutes, then this interval definition returns 1 every 2 minutes.

```
(tag1 > 500) and interval ("30 sec")
```

returns 1 when tag1 > 500 on some 30 second interval since Event Detector was started. (It does *not* mean 30 seconds after tag1 > 500.)

File Functions

The following built-in functions determine whether a specific file exists, and how much free space there is on a disk.

The *file* parameter is the complete DOS pathname, surrounded by quotes.

The *drive* parameter is the letter of the drive.

Table 2.G
File Functions

This function:	returns this value:
FILE_EXISTS("*file*")	TRUE if the specified file exists.
FREE_BYTES(*drive*)	the number of bytes free on the specified drive.

Examples: File Functions

```
FILE_EXISTS("e:\access\dat\activity.000")
```
returns 1 if the file exists or 0 if the file doesn't exist.

```
FREE_BYTES(c)
```
returns the number of bytes available on drive C.

Operator Precedence

There are three rules used to determine the order for evaluating an expression that has more than one operator.

1. When two operators have unequal precedence, the operator with the highest precedence is evaluated first.
2. When two operators have equal precedence, they are evaluated from left to right.
3. To change the normal order of precedence, enclose the statement in parentheses.

Here are the operators from highest to lowest precedence:

Table 2.H
Operator Precedence

Precedent Level	Name	Symbols
1 (Highest)	parentheses	( )
2	Logical negation Bitwise complement	NOT ~
3	multiplication division modulus (remainder) exponent logical and bitwise and bitwise shift right bitwise shift left	* / MOD, % ** AND, && & >> <<
4	addition subtraction logical or bitwise inclusive or bitwise exclusive or the built-in functions	+ – OR, || | ^
5 (lowest)	equal not equal less than greater than less than or equal to greater than or equal to	EQ, = NE, <> LT, < GT, > LE, <= GE, >=

Examples: Operator Precedence

For these examples, tag1 = 5, tag2 = 7 and tag3 =10.

```
(tag1 > tag2) AND (tag1 < tag 3)
```

is evaluated in this sequence:

1. tag1 > tag2 = 0
2. tag 1 < tag3 = 1
3. 0 AND 1 = 0

The expression evaluates to 0.

```
tag1>tag2 AND tag3
```

is evaluated in this sequence:

1. tag2 AND tag3 = 1

2. tag1 > 1 = 1

The expression evaluates to 1.

```
NOT tag1 AND tag2 > tag3 ** 2
```

is evaluated in this sequence:

1. NOT tag1 = 0
2. 0 AND tag2 = 0
3. tag3 ** 2 = 100
4. 0 > 100 = 0

The expression evaluates to 0.

IF-THEN-ELSE

The IF-THEN-ELSE structure allows an expression to make a decision.

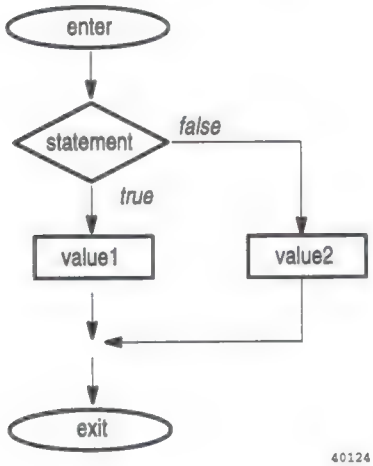
The format of the IF-THEN-ELSE structure is:

IF *statement* THEN *value1* ELSE *value2*

If the *statement* is true then the expression will return *value1*; if the *statement* is false then the expression returns *value2*. Keep in mind that the *statement* is a mathematical equation and that true means a non-zero value, and false means zero.

The IF-THEN-ELSE structure is illustrated below:

Figure 2.3
The IF-THEN-ELSE structure



Nested IF-THEN-ELSE structure

It is common to nest an entire IF-THEN-ELSE structure inside another IF-THEN-ELSE structure.

When nesting IF-THEN-ELSE structures, it is important to remember that an ELSE is associated with the last IF that doesn't have its own ELSE.

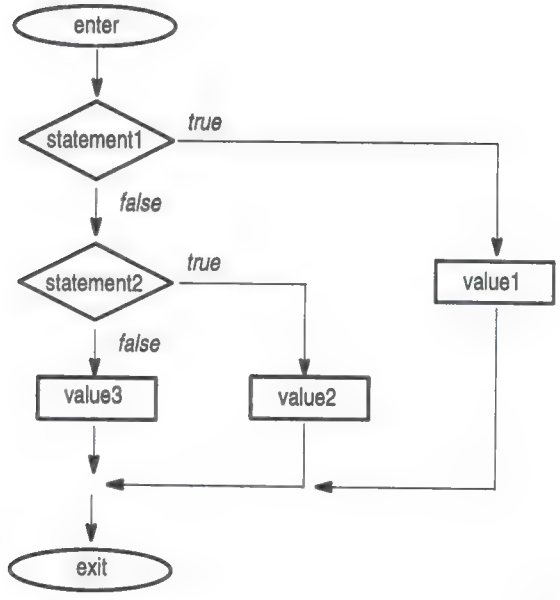
Example 1: Nested IF-THEN-ELSE

This expression:

```
IF (statement1) THEN (value1)
ELSE IF (statement2) THEN (value2)
           ELSE (value3)
```

has this interpretation:

Figure 2.4
Nested IF-THEN-ELSE



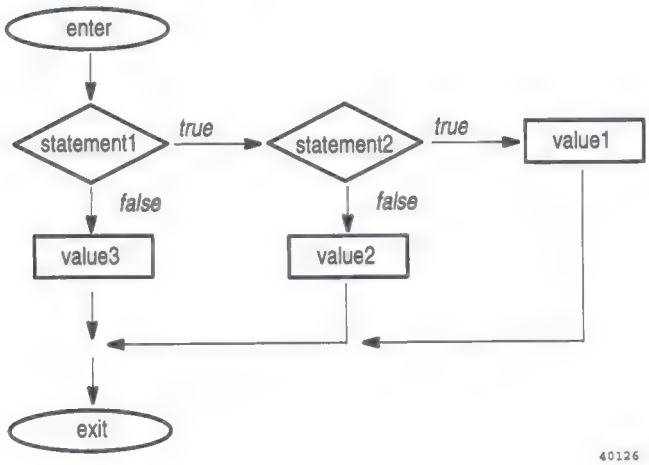
Example 2: Nested IF-THEN-ELSE

This expression:

```
IF (statement1) THEN
           IF (statement2) THEN (value1)
           ELSE (value2)
ELSE (value3)
```

has this interpretation:

Figure 2.5
Nested IF-THEN-ELSE



Comments

Comments begin with an exclamation point and all text following the exclamation point —to the end of that line— is not evaluated.

Example: Comments in ControlView Expressions

In a single line such as:

```
IF a>b THEN c ELSE d ! if/then expression
```

only that part preceding the exclamation point is evaluated.

In several lines such as:

```
IF a>b ! temp a greater than temp b
    THEN c ! shut valve
ELSE d ! do nothing
```

only those parts preceding the exclamation points are evaluated.

Expression Formatting

The event editor ignores tabs, new line characters, and multiple spaces, so a large expression can be condensed to make maximum use of the editor space.

Example: Expression Formatting

The following expression:

```
If (tag1 > tag2) then 0
else if (tag1 > tag 3) then 2
          else 4
```

Could be condensed to the following:

```
If (tag1 > tag2) then 0 else if (tag1 > tag3) then
2 else 4
```

Important: Do not allow tag names, keywords, function names or function arguments to span more than one line.

Appendix A

Event Detector Commands

EVENT

EVENT [*file*] [*/n*]

Calls up the Event Editor and loads an event file. The parameters are:

[*file*]	The name of the event file (without a file extension). If you omit [*file*], the default file, EVENTS, is loaded.
[*/n*]	Turns autoverification off.

EVENTOFF

EVENTOFF

Stops the event detector and unloads the event file.

EVENTON

EVENTON [*file*]

Loads an event file and starts the event detector. The parameter is:

[*file*]	The name of the event file (without a file extension). If you omit [*file*], the default file, EVENTS, is loaded.

Important: Only one event file can be run at a time.

Important: You must load a database before you start event detection.

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

As a subsidiary of Rockwell International, one of the world's largest technology companies — Allen-Bradley meets today's challenges of industrial automation with over 85 years of practical plant-floor experience. More than 11,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines but integrate the manufacturing process, while providing access to vital plant floor data that can be used to support decision-making throughout the enterprise.

With offices in major cities worldwide

WORLD HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (1) 414 382-2000
Telex: 43 11 016
FAX: (1) 414 382-4444

EUROPE/MIDDLE EAST/AFRICA HEADQUARTERS
Allen-Bradley Europe B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel: (31) 2975/43500
Telex: (844) 18042
FAX: (31) 2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen-Bradley (Hong Kong) Limited
Room 1006, Block B, Sea View Estate
28 Watson Road
Hong Kong
Tel: (852) 887-4788
Telex: (780) 64347
FAX: (852) 510-9436

CANADA HEADQUARTERS
Allen-Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel: (1) 519 623-1810
FAX: (1) 519 623-8930

LATIN AMERICA HEADQUARTERS
Allen-Bradley
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (1) 414 382-2000
Telex: 43 11 016
FAX: (1) 414 382-2400

Publication 6190-6.5.10 November 1992
Supersedes Publication 6190-6.5.10,—August 1991

PN 504629804